SLEPc

SLEPc: Scalable Library for Eigenvalue Problem Computations

Jose E. Roman

Departamento de Sistemas Informaticos y Computacion

Universidad Politecnica de Valencia (Spain)

jroman@dsic.upv.es

SLEPc

Problem Definition

- Target problems
  - Large sparse eigenvalue problems
  - Example: discretization of PDEs
- Types of problems
  - Standard: $Ax=\lambda x$
  - Generalized: $Ax=\lambda Bx$
  - Other (SVD, quadratic, …) formulated as one of the above
- Methods
  - “Direct” methods (QR, Jacobi, etc.) are not appropriate
  - Vector iterations
    - Single vector iterations (power, inverse iteration, RQI)
    - Multiple vector iterations (subspace iteration, block RQI)
    - Projection methods (Arnoldi, Lanczos, Jacobi-Davidson)
  - Acceleration techniques: spectral transformations

SLEPc

Acceleration

- Convergence rate is critical for good performance
  - Clustered eigenvalues are a problem
  - Acceleration techniques usually aim at improving separation
- Polynomial filtering
  - Chebychev, least-squares and other polynomials
  - The optimal polynomial is difficult to obtain
- Spectral transformations
  - Origin shift: improvement is very limited
  - Shift-and-invert:
    - Very effective technique, widely used
    - Also allows to find internal eigenvalues
  - More general transforms: Cayley, rational
  - Can be combined with any solution method

$$\boxed{(A-\sigma I)^{-1}x=\mu x}$$

$$\boxed{(A-\sigma B)^{-1}Bx=\mu x}$$

SLEPc

Available Software

- Parallel eigenvalue solvers
  - PARPACK (Sorensen, Lehoucq et al)
    - Implicitly Restarted Arnoldi/Lanczos method
    - Symmetric and nonsymmetric
    - Real and complex problems
  - BLZPACK (Marques)
    - Block Lanczos method
    - For complex problems use HLZPACK
  - PLANSO (Parlett/Wu and Simon)
    - Lanczos with partial reorthogonalization
  - TRLAN (Wu and Simon)
    - Dynamic thick-restart Lanczos algorithm
- Linear systems of equations
  - Direct methods: SuperLU, PSPASES, SPOOLES, (PETSc)
  - Iterative methods: PETSc, Aztec, PIM, Isis++, …

SLEPc Objectives

- Current situation
  - Software for eigenvalue problems available
  - Traditional programming model (Fortran, reverse communication, etc)
  - User has to be aware of details of parallelism
  - Usually has to be combined with software for the solution of linear systems of equations

- With SLEPc
  - More modern programming paradigm
    - Object oriented
    - Built-in programming tools: debugging, visualization, etc.
    - Details of parallelism hidden in lower abstraction levels
  - Very easy to implement an eigensolver
    - The user still has control over all the details of solution method
  - All the functionality of PETSc available

SLEPc

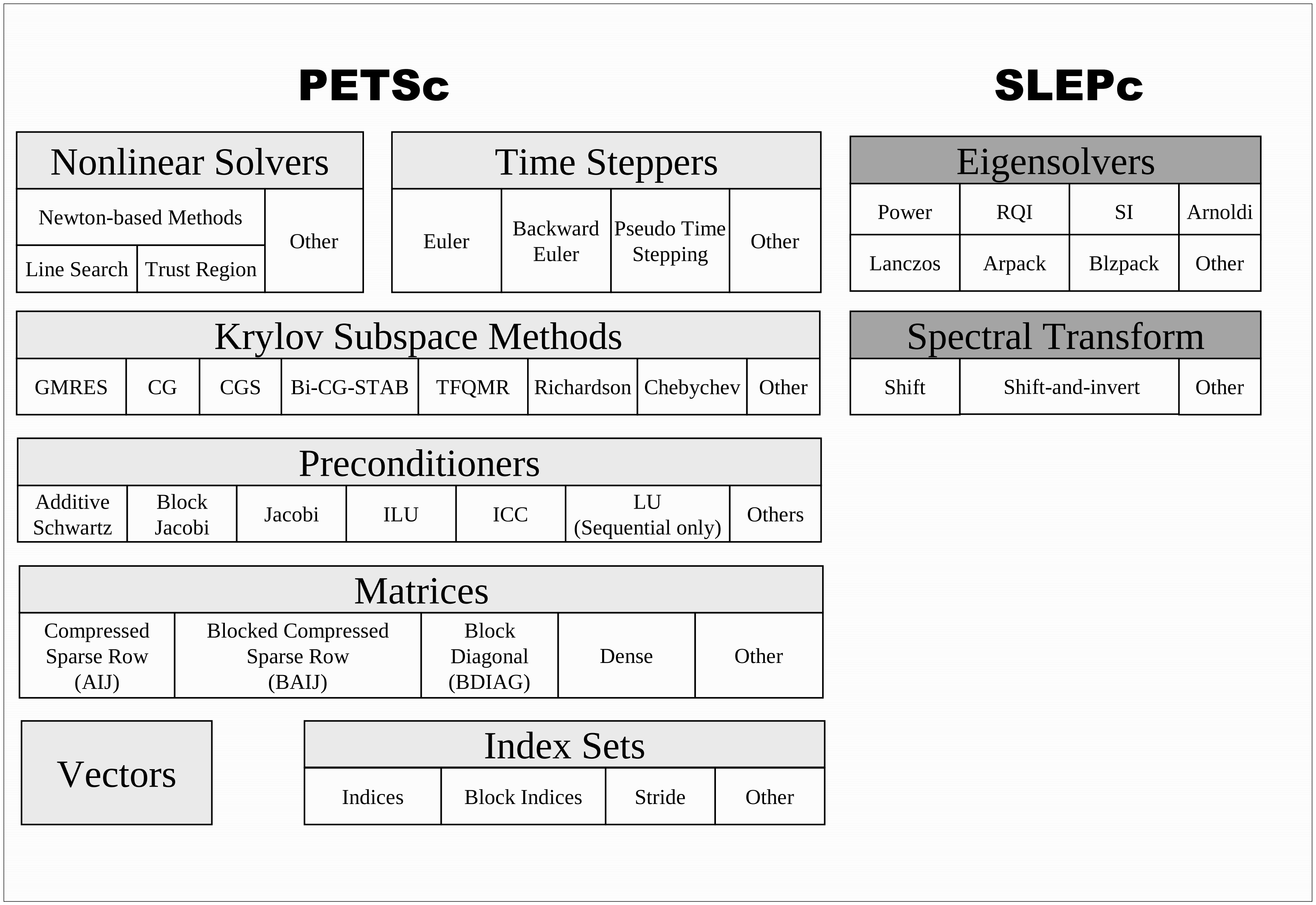
What is SLEPc?

- SLEPc: Scalable Library for Eigenvalue Problem Computations
- A new library
  - Aim: solution of large scale sparse eigenvalue problems
  - Can be considered an extension of PETSc
  - Developed by HPNC group in Valencia (Spain)
- Properties
  - Freely available (and supported) research code
  - Hyperlinked documentation and manual pages for all routines
  - Many tutorial-style examples
  - (Support via email)
  - Usable from Fortran 77/90, C, and C++
  - Portable to any parallel system supporting MPI
  - Good parallel performance
  - Extensible

… the same way as PETSc

SLEPc

Numerical Components



SLEPc

SLEPc objects (1)

EPS: Eigenvalue Problem Solver

- Solvers for
  - Standard and generalized
  - Real and complex arithmetic
  - Hermitian and non-Hermitian
- Main functions
  - **`EPSCreate, EPSDestroy, EPSView, EPSSetOperators, EPSSetInitialVector, EPSSetUp, EPSSolve`**
- The user can
  - Select a solver
  - Specify various parameters:
    - nev: number of eigenvalues
    - ncv: dimension of the subspace (number of basis vectors)
    - tolerance, max iterations, portion of the spectrum
    - orthogonalization technique (CGS, MGS, IR, DGKS, other)
    - whether to compute eigenvectors or not
  - Via procedural or command line interface

SLEPc

SLEPc objects (2)

ST: Spectral Transformation

- How it works
  - Solvers apply the “operator” to a vector
  - The “operator” is different depending on the type of ST
  - Linear systems are handled via a SLES object
  - After convergence, eigenvalues have to be transformed back appropriately

	Standard	Generalized
none	A	$B^{-1}A$
shift	$A+\sigma I$	$B^{-1}A+\sigma I$
sinvert	$(A-\sigma I)^{-1}$	$(A-\sigma B)^{-1}B$

- Main functions
  - `STCreate, STDestroy, STView, STSetUp, STApply`
- The user can
  - Select the type of transformation
    - Type `shell` also available for user-defined transformations
  - Specify various parameters: the value of the shift (σ)
  - In `sinvert` also the linear system solver and corresponding options

SLEPc

Basic Eigensolver Code

```
EPS   eps;              /*  eigensolver context   */
Mat   A;                /*  matrix  */
Vec   *x;               /*  basis vectors   */
Scalar *kr,*ki;         /*  eigenvalues */

MatCreate(MPI_COMM_WORLD,n,n,N,N,&A);
MatSetFromOptions(A);
/* assemble matrix */

EPSCreate(MPI_COMM_WORLD,&eps);
EPSSetOperators(eps,A,PETSC_NULL);
EPSSetFromOptions(eps);
EPSSolve(eps,&its);
EPSGetConverged(eps,&nconv);
EPSGetSolution(eps,&kr,&ki,&x);
EPSComputeError(eps,error);
```

SLEPc

Power Method

- **`power`**
  - Power method
  - Deflation for computing more than one eigenpair
  - Combined with shift-and-invert is equivalent to inverse iteration
- **`rqi`**
  - Rayleigh Quotient Iteration
  - Only implemented for one eigenpair
- Examples

```
ex1 -eps_type power -eps_tol 1e-8 -eps_monitor
ex1 -eps_type power -eps_nev 6
ex1 -eps_type power -st_type shift -st_shift 0.5
ex1 -eps_type power -st_type sinvert -st_shift 2000
ex1 -eps_type rqi -eps_monitor_values
```

SLEPc

Subspace Iteration

- **`subspace`**
  - Subspace Iteration method
  - Non-Hermitian projection
  - Deflation by locking converged eigenpairs

- Examples

```
ex1 -eps_type subspace -eps_nev 1 -eps_ncv 12
ex1 -eps_type subspace -eps_mgs_orthog
ex1 -eps_type subspace -eps_plot_eigs -draw_pause 10
ex1 -eps_type subspace -st_type sinvert -st_shift 1
    -sinv_ksp_type gmres -sinv_pc_type sor
    -sinv_pc_sor_omega 1.2
ex1 -eps_type subspace -none_ksp_type cg
    -none_pc_type jacobi -none_ksp_tol 1e-5
ex1 -eps_type subspace -eps_view
```

SLEPc

Arnoldi, Lanczos and Wrappers

- **`arnoldi`**
  - Arnoldi method
  - Explicit restart and deflation
- **`lanczos`**
  - (Hermitian) Lanczos method
  - Full reorthogonalization
- Wrappers
  - **`arpack, blzpack, planso, trlan`**
  - Specific options for some of them, e.g. `-eps_blzpack_block_size`
  - Also **`lapack`** for validation purposes
  - When installing SLEPc the user specifies which of this packages are available

SLEPc

Examples

Category	Filename	Description
1	ex1.c	1-D Laplacian, standard symmetric eigenproblem
3	ex1f.F	Fortran equivalent of ex1.c
1	ex2.c	2-D Laplacian, standard symmetric eigenproblem
4	ex3.c	2-D Laplacian, matrix-free version
1	ex4.c	Matrix loaded from a file, standard problem
1	ex5.c	Markov model of a random walk in a triangular grid
9	ex6f.F	Ising model for ferromagnetic materials
1	ex7.c	Matrices loaded from a file, generalized problem
6	ex8.c	Grcar matrix, Singular Value Decomposition
1	ex9.c	Brusselator model, standard nonsymetric w/blocks

Also tests with NEP collection (math.nist.gov/MatrixMarket)

SLEPc

The Future

- First version:
  - Will probably be released in Nov or Dec 2001
  - Version numbering will probably be consistent with PETSc
    - Therefore: SLEPc 2.1.0
  - Will contain
    - Power, Subspace iteration, RQI, Arnoldi, Lanczos
    - Also wrappers to Arpack, Blzpack, Planso, Trlan, Lapack
    - Shift and shift-and-invert spectral transformations
- What is next?
  - Close collaboration with PETSc team
  - More methods (Non-Hermitial Lanczos, Jacobi-Davidson, …)
  - Other spectral transformations or acceleration techniques
  - Further testing with several case studies
- Open to external collaboration
  - Researchers who want to experiment with new methods
  - Users with interesting applications

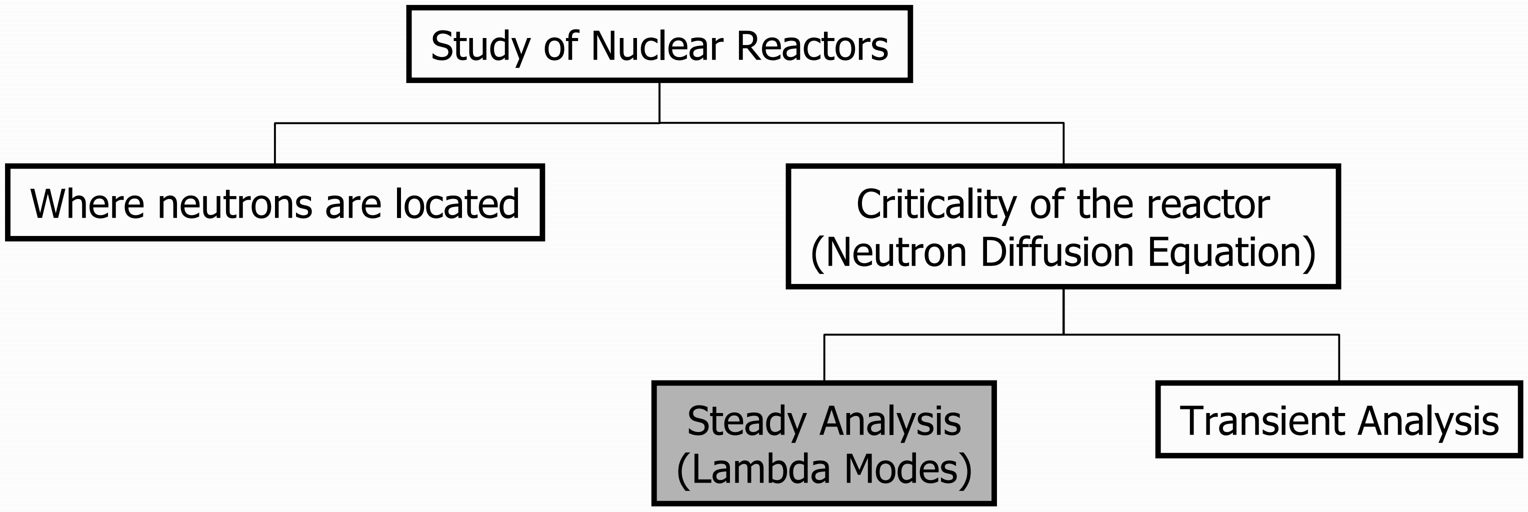
Lambda Modes

- Nuclear reactor analysis
- The Lambda Modes equation
- Modeling the reactor
- Solution strategy
- Implementation with SLEPc
- Preliminary performance results

SLEPc

Introduction

- Context: Security analysis in nuclear reactors
  - The main aim is to improve security
  - Also reduction of production costs can be pursued
  - Engineering companies demand tools for detailed analysis
  - This analysis has evolved to 3D methodologies


Study of Nuclear Reactors
Where neutrons are located
Criticality of the reactor (Neutron Diffusion Equation)
Steady Analysis (Lambda Modes)
Transient Analysis

- Lambda Modes analysis
  - eigenvalues and eigenvectors of time-independent neutron diffusion equation of a nuclear reactor

Criticality

Criticality: depends on how many of the free neutrons from each fission, on average, hits another U-235 nucleus and causes it to split:

- Exactly one: the mass is **critical**
  - The mass will exist at a stable temperature
- Less than one: the mass is **subcritical**
  - Eventually, induced fission will end in the mass
- More than one: the mass is **supercritical**
  - It will heat up
  - In a nuclear reactor, the reactor core needs to be slightly supercritical so that plant operators can raise and lower the temperature of the reactor
  - The control rods give the operators a way to absorb free neutrons so that the reactor can be maintained at a critical level

SLEPc

The Physical Problem

- Lambda Modes equation
  - Derived from time-independent neutron diffusion equation
  - Multigroup approach: neutrons are grouped in energy intervals. With two energy groups (fast and thermal):

$$L\phi_i = \frac{1}{\lambda_i} M\phi_i$$

$$L = \begin{bmatrix} -\nabla(D_1\nabla) + \Sigma_{a1} + \Sigma_{12} & 0 \\ -\Sigma_{12} & -\nabla(D_2\nabla) + \Sigma_{a2} \end{bmatrix}$$

$$M = \begin{bmatrix} v_1\Sigma_{f1} & v_2\Sigma_{f2} \\ 0 & 0 \end{bmatrix} \qquad \phi_i = \begin{bmatrix} \phi_{f_i} \\ \phi_{t_i} \end{bmatrix}$$

SLEPc

The Algebraic Problem

- Discretization: Nodal Collocation, with Legendre polynomials

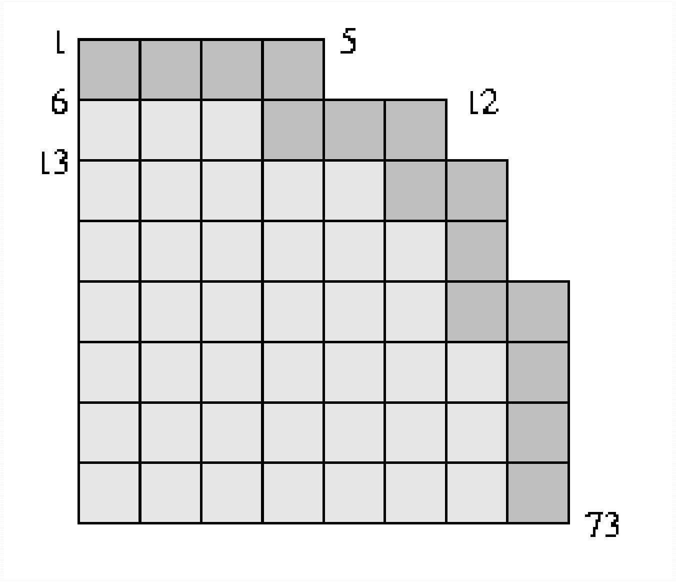
$$\boxed{\begin{gathered} L\psi_i = \frac{1}{\lambda_i} M\psi_i \\ \begin{bmatrix} L_{11} & 0 \\ -L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} \psi_{1_i} \\ \psi_{2_i} \end{bmatrix} = \frac{1}{\lambda_i} \begin{bmatrix} M_{11} & M_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \psi_{1_i} \\ \psi_{2_i} \end{bmatrix} \end{gathered}}$$

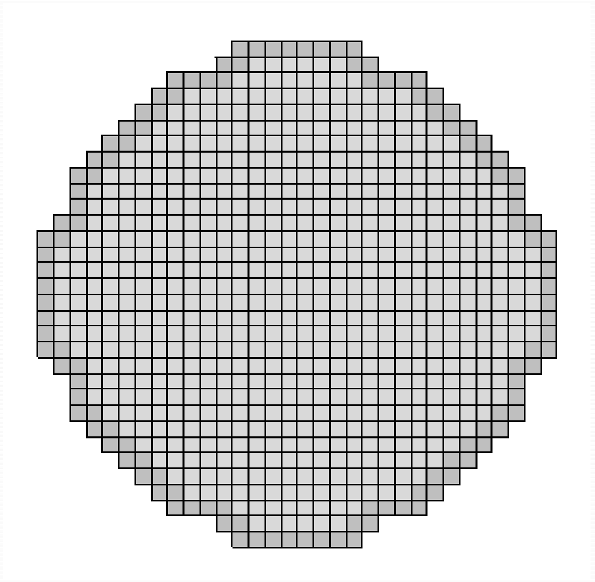
- Reduced size eigenproblem

$$\boxed{\begin{gathered} A\psi_{1_i} = \lambda_i \psi_{1_i} \\ A = {L_{11}}^{-1}(M_{11} + M_{12}{L_{22}}^{-1}L_{21}) \end{gathered}}$$

SLEPc

Nodalization





2D Case

All axial planes are considered equal

¼ symmetry

3D Case

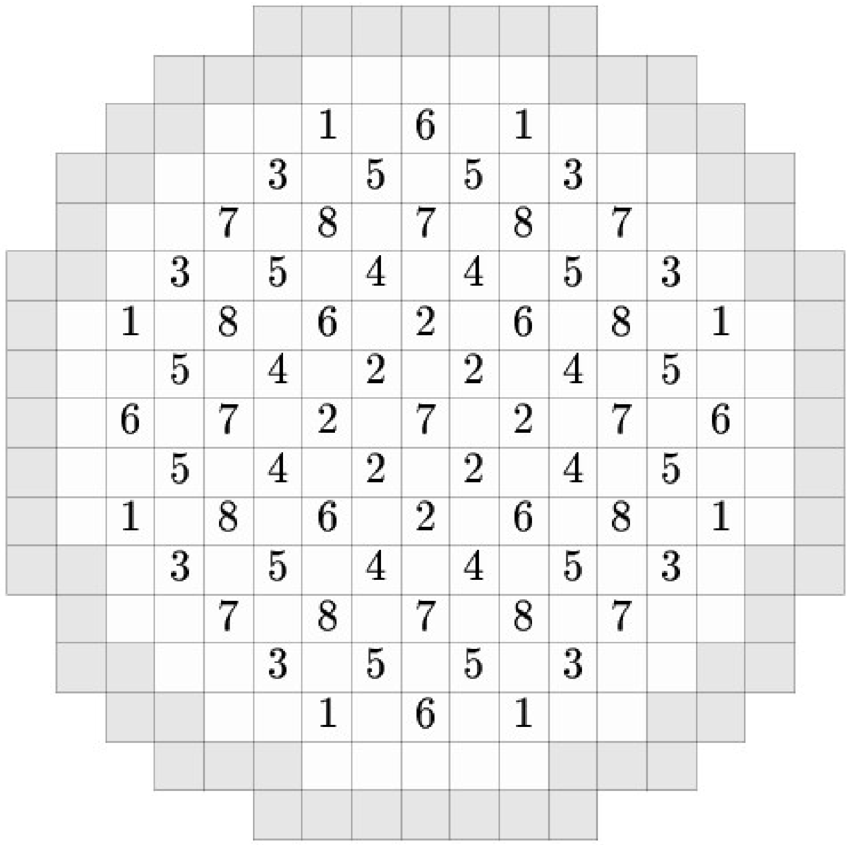
Different axial planes

No symmetries

Other details such as control rods

SLEPc

Control Rods



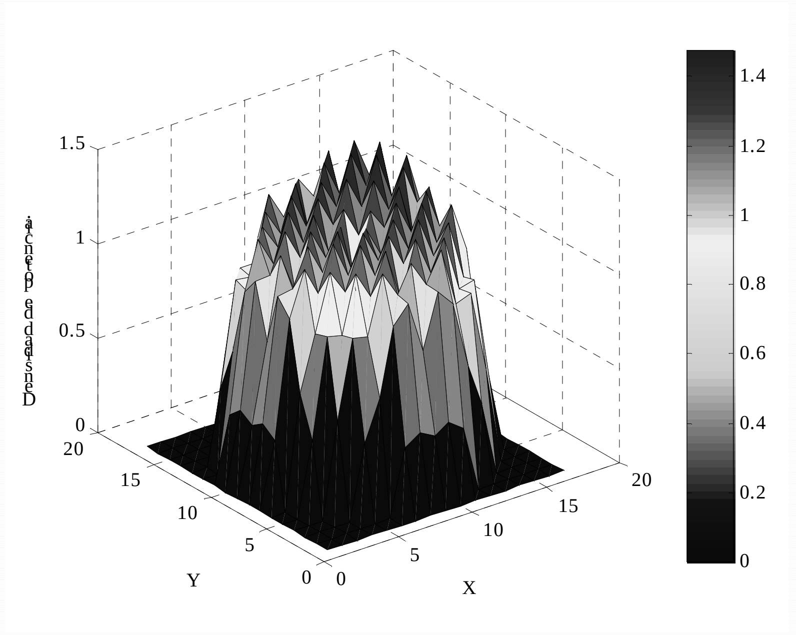
Bank	No. of rods	Purpose
1	8	Security
2	8	Security
3	8	Security
4	8	Security
5	12	Regulation
6	12	Regulation
7	9	Regulation
8	8	APSR

The user can specify the position of each control rod bank (e.g. bank 7 inserted 20%)

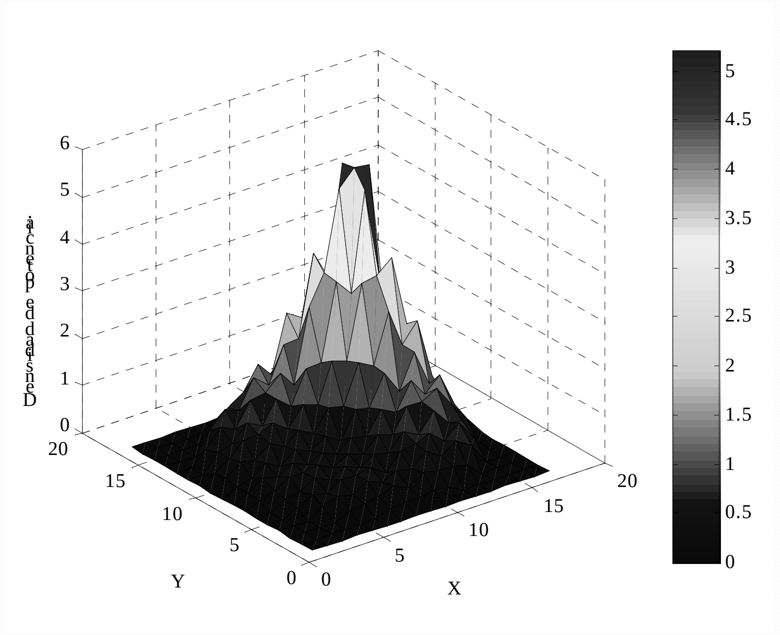
SLEPc

Analysis of Results

- Physical interpretation of results
  - Eigenvalues inform about criticality of the reactor
  - Eigenvectors inform about distribution of power density



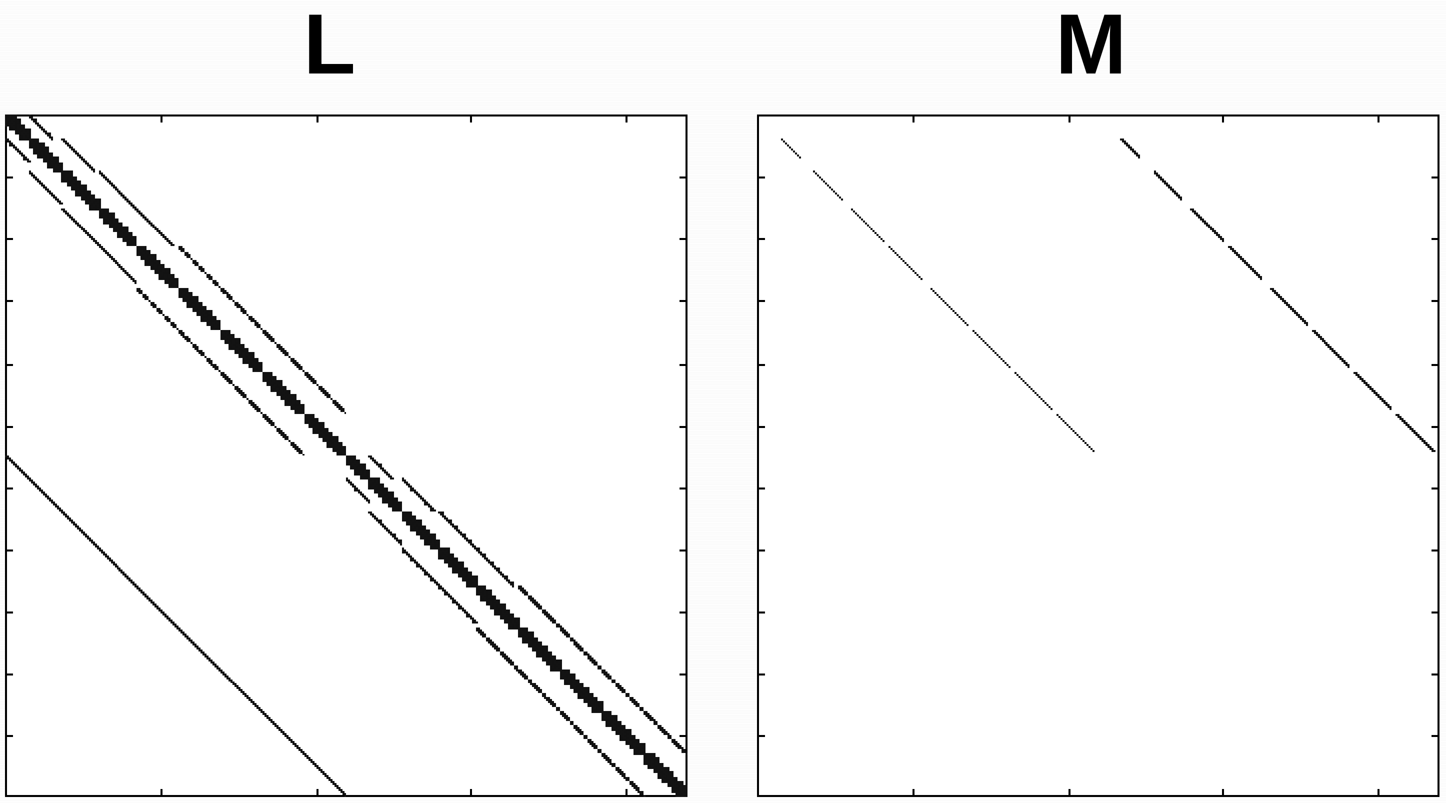
normal conditions



abnormal conditions

SLEPc

Nonzero Pattern (2D)



L_{11}, L_{22} : symmetric matrices, (almost always) positive definite
M_{11}, M_{12}, L_{21} : diagonal matrices

SLEPc

Solution Strategy

- Large eigenvalue problem
  - Size in 3D models range from 50000 to 1 million depending on degree of polynomial
  - “Sparse” methods are preferred
- Several approaches
  - Full eigensystem
  - Reduced generalized eigensystem
  - Reduced standard eigensystem
- In the latter case, matrix vector products are done without forming the matrix explicitly

$$A\psi_{1_i} = \lambda_i \psi_{1_i}$$
$$A = L_{11}^{-1}(M_{11} + M_{12} L_{22}^{-1} L_{21})$$

$y=Ax$ ⇨

$w_1=M_{11}x$
$w_2=L_{21}x$
Solve $L_{22}w_3=w_2$
$w_4=w_1+M_{12}w_3$
Solve $L_{11}y=w_4$

SLEPc

Main Program

```
#include "slepceps.h"
int main(int argc,char** argv)
{
  SlepcInitialize(&argc,&argv,(char*)0,help);
  LambdaGetOptions(&reac,&dpol);
  MatLambdaCreate(reac,dpol,&A);
  EPSCreate(comm,&eps);
  EPSSetOperators(eps,A,PETSC_NULL);
  EPSSetFromOptions(eps);
  EPSSolve(eps,&its);
  EPSGetConverged(eps,&nconv);
  EPSGetSolution(eps,&kr,&ki,&x);
  EPSComputeError(eps,error);
  MatDestroy(A);
  EPSDestroy(eps);
  SlepcFinalize();
}
```

SLEPc

Matrix Generation

```
typedef struct {
  SLES       L11, L22;
  Vec        w, L21, M11, M12;
} CTX_LAMBDA;

int MatLambdaCreate(Reactor reac,int dpol,Mat *A)
{
  CTX_LAMBDA      *ctx;
  /* generate M with appropriate ordering */
  SLESCreate(comm,&ctx->L11);
  SLESSetOperators(ctx->L11,M,M,flag);
  SLESGetKSP(ctx->L11,&ksp);CHKERRQ(ierr);
  KSPSetType(ksp,KSPCG);CHKERRQ(ierr);
  SLESGetPC(ctx->L11,&pc);CHKERRQ(ierr);
  PCSetType(pc,PCJACOBI);CHKERRQ(ierr);
  SLESSetFromOptions(ctx->L11); CHKERRQ(ierr);
  MatCreateShell( comm, n, n, N, N, (void*)ctx, A );
  MatShellSetOperation(*A,MATOP_MULT,MatLambda_Mult);
}
```

Repeated for L22

SLEPc

Matrix-vector Product

```
int MatLambda_Mult( Mat A, Vec x, Vec y )
{
  CTX_LAMBDA *ctx;
  int        its, ierr;
  Scalar     done = 1.0;

  MatShellGetContext( A, (void**)&ctx );
  VecPointwiseMult( ctx->L21, x, ctx->w );
  SLESSolve( ctx->L22, ctx->w, y, &its );
  VecPointwiseMult( ctx->M12, y, ctx->w );
  VecPointwiseMult( ctx->M11, x, y );
  VecAXPY( &done, y, ctx->w );
  SLESSolve( ctx->L11, ctx->w, y, &its );
  PetscFunctionReturn(0);
}
```

SLEPc

Preliminary Performance Results

- Timings in a SGI Origin 2100 system (8 proc)
- No renumbering of nodes
- Block size n=83376

p	T_p	S_p	E_p (%)
1	218.64	1.00	100
2	147.43	1.48	74
4	58.40	3.47	94
6	46.45	4.71	78

- Block size n=208440

p	T_p	S_p	E_p (%)
1	1464.44	1.00	100
2	905.71	1.62	81
4	479.87	3.05	76
6	327.28	4.47	75